Performance and Bandwidth Optimization for Biological Sequence Alignment

Laiq Hasan Zaid Al-Ars Mottaqiallah Taouil Koen Bertels
Computer Engineering Lab, Delft University of Technology,
Mekelweg 4, 2628 CD, Delft, The Netherlands
E-mail: L.HASAN@TUDELFT.NL

***Abstract*— Sequence alignment is an essential, but compute-intensive application in Bioinformatics. Hardware implementation speeds up this application by exploiting its inherent parallelism, where the performance of the hardware depends on its capability to align long sequences. In hardware terms, the length of a biological query sequence that can be aligned against a database sequence depends on the number of *Processing Elements (PEs)* available, which in turn depends on the amount of available hardware resources. In addition, the amount of available bandwidth to transfer the data processed by these PEs plays a significant role in defining the maximum performance. In this paper, we carry out a detailed performance and bandwidth analysis for biological sequence alignment and formulate theoretical performance boundaries for various cases. Further, we optimize the performance gain and memory bandwidth requirements and develop generalized equations for this optimization.**

***Index Terms*— Sequence Alignment, Smith-Waterman Algorithm, FPGAs, Performance Gain, Memory Bandwidth**

I. Introduction

Sequence alignment is a computationally intensive and bandwidth hungry activity in Bioinformatics [1], [2]. Various methods are available for local and global sequence alignments [3]. Methods like BLAST [4], FASTA [5] and HMMER [6] are fast, but they are based on heuristics and do not guarantee an optimal alignment. Based on *dynamic programming (DP)* [7], the *Smith-Waterman (S-W)* algorithm [8] is a method that finds an optimal local sequence alignment between two DNA or protein sequences, i.e. the *query sequence* of length N_q and the *database sequence* of length N_s. The S-W algorithm has a limitation that it requires an extensive computation time, when used for aligning long sequences. Therefore, it is desired to come up with an efficient and fast hardware based design for the S-W algorithm.

In hardware, the S-W algorithm is most often implemented as a linear systolic array [9]. The performance of such arrays depends on the number of PEs and the available memory bandwidth. The more the PEs, the longer the query sequences that can be compared and aligned against the database sequences in a specific amount of time. The quantity of PEs that can be placed, routed and utilized, in turn depends on the availability of the amount of hardware resources. Current *Field Programmable Gate Array (FPGA)* technology offers abundant hardware resources, sufficient for fitting large number of PEs. Therefore, for several applications including sequence alignment, the maximum performance is limited by the available memory bandwidth. The more the memory bandwidth, the more the overall performance gain. Full scale implementation, i.e. utilizing the maximum available PEs on a given platform, improves the performance, but increases the bandwidth requirement to transfer the generated output data. Work has been done on accelerating the S-W algorithm in hardware [9], [10], [11], [12], [13], but no tangible effort has been made to optimize the bandwidth requirement for transferring the huge data generated by hardware designs that are capable of handling long query sequences.

In this paper, we present the performance and bandwidth analysis for hardware based S-W algorithm and formulate theoretical performance boundaries for various cases. Further, generalized equations are developed for optimizing the performance gain and memory bandwidth requirements.

The remainder of the paper is organized as follows:

Section II presents the background and motivation behind the work. Section III presents the theoretical performance boundaries. Section IV presents an analysis for the case, when performance is limited by the computational resources. Section V presents an analysis for the case, when performance is limited by the bandwidth. Section VI gives the performance and bandwidth optimization. Section VII concludes the paper.

II. Background and Motivation

For the S-W local alignment, a matrix H is used to keep track of the degree of similarity between the two sequences to be aligned, i.e. the query sequence and the database sequence. Each element of the matrix, with a row index i and a column index j, is calculated according to the following equation,

$$H_{i,j} = \max \begin{cases} 0 \\ H_{i-1,j-1} + S_{i,j} \\ H_{i-1,j} - d \\ H_{i,j-1} - d \end{cases} \quad (1)$$

where $S_{i,j}$ is the similarity score of comparing the residues of the two sequences and d is the penalty for a mismatch. A detailed description of the S-W algorithm and its data dependencies is given in [13]. Table I shows a sample H

matrix for aligning two sequences of m characters each. If the precision of the aligned output data is 16 bits wide [14], then, for $m = 500$ (74% of sequences in Swiss-Prot are of length ≤ 500 [15]), the total amount of data that needs to be stored in memory is, $500 \times 500 \times 16 = 4$ Mbits. This amount increases with the increasing length of the query and database sequences.

TABLE I

H MATRIX FOR ALIGNING SEQUENCES OF m CHARACTERS EACH

	A	**C**	...	...	**T**	**G**
G	$H_{1,1}$	$H_{1,2}$	...	...	$H_{1,m-1}$	$H_{1,m}$
A	$H_{2,1}$	$H_{2,2}$	...	...	$H_{2,m-1}$	$H_{2,m}$
...	...	...	...	...	...	...
...	...	...	...	...	...	...
T	$H_{m-1,1}$	$H_{m-1,2}$	...	...	$H_{m-1,m-1}$	$H_{m-1,m}$
C	$H_{m,1}$	$H_{m,2}$	...	...	$H_{m,m-1}$	$H_{m,m}$

In practice, e.g. FPGA implementations, a large number of PEs is required to align long sequences. The larger the number of PEs, the longer the query sequences that can be aligned against the database sequences and the better the performance. When all the PEs are simultaneously active, the bandwidth required to store the resultant output data increases with the increasing array length. Hence, the on-chip local *Block RAM (BRAM)* becomes very limited for storing all the intermediate values and can only be used as a buffer that transfers the data to an off-chip main memory, e.g. the *Double Data Rate (DDR)* RAM. Figure 1 gives a block diagram description of such a system. Thus, the overall performance of the hardware system not only depends on the availability of computational resources, i.e. the number of PEs, but also on the bandwidth of the main memory (B_{main}). Both the issues are elaborated in Sections IV and V respectively. Implementation details of the hardware system are skipped, as the main goal of the paper is to come up with theoretical performance boundaries and a subsequent performance and bandwidth optimization.

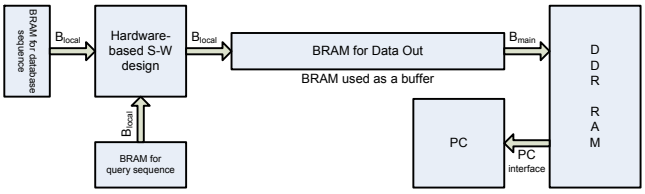

Fig. 1. Block diagram description of an FPGA based design for aligning long sequences

III. THEORETICAL PERFORMANCE BOUNDARIES

The total *execution time (T_{exec})* for the hardware based S-W design is given by the following equation,

$$T_{exec} = T_{compute} + T_{access} \tag{2}$$

where $T_{compute}$ is the total computation time and T_{access} is the total time to load/store data in memory.

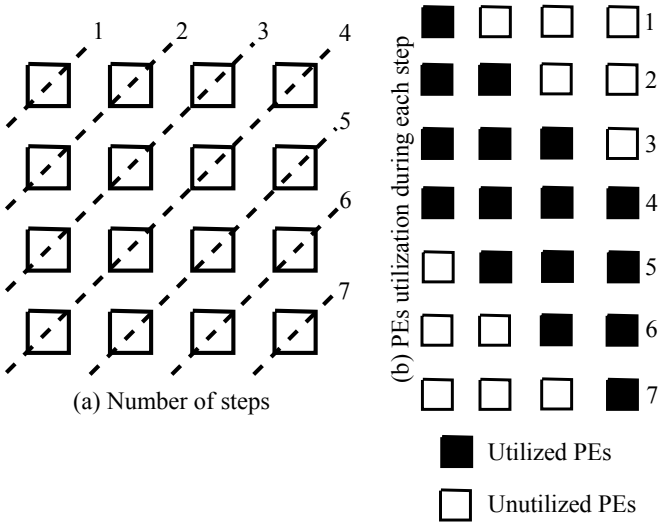

Fig. 2. Number of steps and PEs utilization during each step for a case, where $N = N_q = N_s$

For a square scoring matrix, i.e. when $N_q = N_s$, the data flow is as shown in Figure 2(a). Since the elements within each anti-diagonal are processed in parallel, therefore, the computation time is given by,

$$T_{compute} = (2N - 1) \times T_{PE} \tag{3}$$

where N is the number of PEs, such that, the total number of steps needed for the entire computation is $(2N - 1)$. T_{PE} is the computation time for 1 step, such that,

$$T_{PE} = C_{PE} \times T_{cycle}$$

where C_{PE} is the number of cycles consumed by 1 PE and T_{cycle} is the time for 1 cycle. T_{PE} is equal to the computation time for 1 PE, as the PEs utilized during each step are processed in parallel. For the given example in Figure 2,

$$N = N_q = N_s = 4 \Rightarrow \text{Total number of steps} = 2N - 1 = 7$$

In Figure 2(b), each row represents the number of PEs available in each step, whereas, the solid black cells represent the number of PEs utilized, such that,

$$\text{Utilization ratio} = \frac{\text{PEs utilized}}{\text{PEs available}} = \frac{N_q^2}{N \times N_{steps}} \tag{4}$$

where $N_{steps} = 2N - 1$

The total time to transfer data to the main memory is given by,

$$T_{access} = \frac{D_{main}}{B_{main}} \tag{5}$$

where D_{main} is the total data that needs to be stored in the main memory and B_{main} is the bandwidth of the main memory in bits/sec, such that,

$$D_{main} = 16N_s \times N_q$$

where the precision of each output is 16 bits wide. When $N_q = N_s = N$, the total data becomes,

$$D_{main} = 16N^2 \tag{6}$$

Substituting Equation 6 in Equation 5,

$$T_{access} = \frac{16N^2}{B_{main}} \tag{7}$$

Substituting Equation 3 and 7 in Equation 2,

$$T_{exec} = \frac{(2N-1) \times T_{PE} \times B_{main} + 16N^2}{B_{main}} \tag{8}$$

Now, the overall performance is given by the ratio of the total matrix fill operations to the total execution time, i.e.

$$\text{Overall performance} = P_{overall} = \frac{\text{Total operations}}{T_{exec}}$$

$$P_{overall} = \frac{N^2}{\frac{(2N-1)\times T_{PE}\times B_{main}+16N^2}{B_{main}}}$$

$$P_{overall} = \frac{N^2 \times B_{main}}{(2N-1) \times T_{PE} \times B_{main} + 16N^2} \tag{9}$$

If the number of PEs is less than the length of the query sequence, then we need to partition the query sequence, so that, the computation takes place in k passes, where $k \geq 1$ is an integer. In this case the total time becomes,

$$T_{total} = k \times T_{exec}, \text{ where } k = \left\lceil \frac{N_q}{N} \right\rceil$$

and the total operations become $k \times N^2$.

Parameters that can limit the performance of the hardware based S-W design are:

- The amount of available hardware computational resources, i.e. the number of PEs
- The memory bandwidth

In the following two sections, we present performance analysis based on these limitations.

IV. Performance limited by the computational resources

Assume that we have infinite bandwidth and the performance is only limited by the computational resources, then,

$$P_{compute} = f(f_{op}, N)$$

where $P_{compute}$ is the performance limited by the computational resources, f_{op} is the operating frequency and N is the number of PEs. In this case, $T_{access} \cong 0$, so,

$$T_{exec} = T_{compute} = N_{steps} \times T_{step} \tag{10}$$

where N_{steps} is the number of anti-diagonals and T_{step} is the time taken by each anti-diagonal. Now, performance limited by the computational resources is given by,

$$P_{compute} = \frac{N_q \times f_{op}}{C_{PE}} \times \text{Utilization ratio} \tag{11}$$

where C_{PE} is the number of cycles consumed by 1 PE and *Utilization ratio* is the ratio of *utilized* to *available* computational resources. Performance may also be given by,

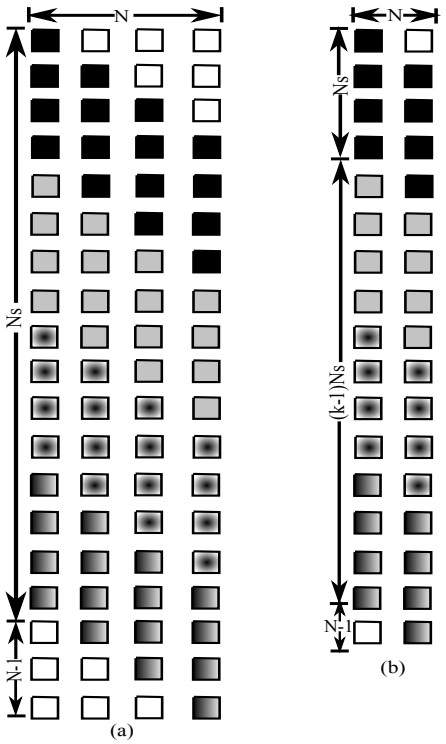

Fig. 3. Number of steps and PEs utilization for (a) $N = N_q < N_s$ and (b) $N < N_s \leq N_q$

$$P_{compute} = \frac{\text{Total operations}}{T_{exec}} \tag{12}$$

Figure 3(a) depicts a case, where the number of PEs is the same as the length of the query sequence i.e. $N = N_q = 4$ and the length of the database sequence is larger than the number of PEs i.e. $N_s = 16$. Equation 10 for this case becomes,

$$T_{exec}|_{N=N_q<N_s} = (N_s + N - 1) \times T_{PE} \tag{13}$$

The utilization ratio is given by Equation 14, where the utilization is dependent on the $\frac{N-1}{N_s}$ term. The lower the $\frac{N-1}{N_s}$ term, the more efficient the hardware resource utilization.

$$\begin{aligned} \text{Utilization ratio}|_{N=N_q<N_s} &= \frac{N_s \times N}{(N_s + N - 1) \times N} \\ &= \frac{1}{1 + \frac{N-1}{N_s}} \end{aligned} \tag{14}$$

Figure 3(b) depicts the case, when $N < N_s \leq N_q$. In this case, we partition N_q into k parts, where the length of each part is $N_q^{'}$, such that, $N_q^{'} = N$. In other words, we scale N_q down to N and perform multiple (k) passes instead, where $k = \left\lceil \frac{N_q}{N_q^{'}} \right\rceil$. This approach is referred to as *Query Sequence Partitioning (QSP)*. For the given example,

$$N_q = 8,\ N_s = 4,\ N_q^{'} = N = 2 \ \text{ and } \ k = \left\lceil \frac{8}{2} \right\rceil = 4$$

The execution time for this case becomes,

$$T_{exec}|_{k>1,\ N<N_s\leq N_q} = (N_s + (k-1)\times N_s + N - 1)\times T_{PE}$$
$$= (k\times N_s + N - 1)\times T_{PE} \quad (15)$$

The utilization ratio for this case is given by Equation 16, where the utilization is dependent on the $\frac{N-1}{k\times N_s}$ term. The lower the $\frac{N-1}{k\times N_s}$ term, the more efficient the hardware resource utilization.

$$\text{Utilization ratio}|_{k>1,\ N<N_s\leq N_q} = \frac{k\times N_s\times N}{(k\times N_s + N - 1)\times N}$$
$$= \frac{1}{1+\frac{N-1}{k\times N_s}} \quad (16)$$

Table II presents the corresponding calculated values of T_{exec} in *microseconds (μsec)* for various combinations of k and N, as per Equation 15, where $T_{PE} = 10\ ns$ and $N_q = N_s = 500$. The inputs are taken from Swiss-Prot, where 74% of the sequences are of length ≤ 500. The table shows that if we have the same number of processing elements as the length of the query sequence, i.e. $N = N_q = 500$, then the computation takes place in one pass, i.e. $k = 1$, which completes in $9.99\ \mu sec$. But if the number of PEs available are half the length of the query sequence, i.e. $N = \frac{1}{2}N_q$, then the computation completes in two passes, i.e. $k = 2$, that takes $12.49\ \mu sec$. Note that by halving the number of PEs, the execution time is increasing only by 25%, however, using half of the resources requires half of the bandwidth for data transfer.

TABLE II
EXECUTION TIME (T_{exec}) IN μsec FOR VARIOUS COMBINATIONS OF k AND N, AS PER EQUATION 15

$N,\ k$	T_{exec}	$N,\ k$	T_{exec}	$N,\ k$	T_{exec}
$N=1$, $k=500$	2500	$N=10$, $k=50$	250.09	$N=100$, $k=5$	25.99
$N=2$, $k=250$	1250	$N=20$, $k=25$	125.19	$N=125$, $k=4$	21.24
$N=4$, $k=125$	625	$N=25$, $k=20$	100.24	$N=250$, $k=2$	12.49
$N=5$, $k=100$	500	$N=50$, $k=10$	50.49	$N=500$, $k=1$	9.99

Figure 4 shows T_{exec} versus the number of PEs (N) curve, limited by the computational resources. The curve shows that T_{exec}, as per Equation 15, decreases with the increasing number of PEs (N).

V. PERFORMANCE LIMITED BY THE BANDWIDTH

Assume that we have infinite computational resources, i.e. zero computation time, then,

$$T_{exec} = T_{access},\ \text{as},\quad T_{compute} \cong 0$$

Applying Equation 7,

$$T_{exec} = T_{access} = \frac{16N^2\ (bits)}{B_{main}\ (Mbps)} = \frac{2N^2\ (Bytes)}{B_{main}\ (MBps)} \quad (17)$$

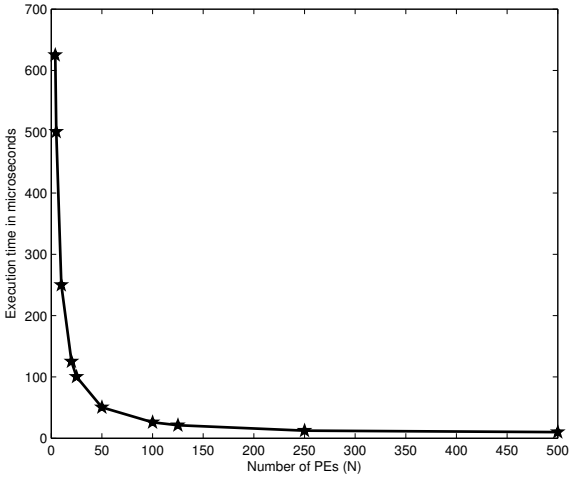

Fig. 4. T_{exec} vs N curve, limited by the computational resources, where T_{exec} decreases with the increasing N values

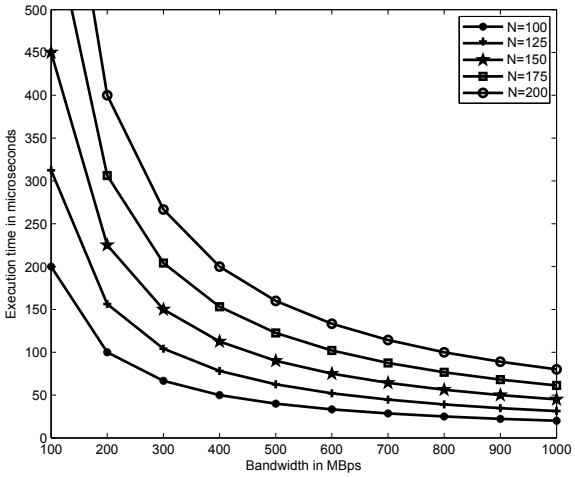

Fig. 5. T_{exec} vs bandwidth curves, where T_{exec} decreases with the increasing amount of bandwidth, for particular number of PEs (N)

where $Mbps$ is the bandwidth in *Mega bits per second* and $MBps$ is the bandwidth in *Mega Bytes per second.*

Now, performance limited by the bandwidth is,

$$P_{bandwidth} = \frac{\text{Total operations}}{T_{exec}}$$

$$P_{bandwidth} = \frac{N^2}{\frac{2N^2}{B_{main}}} = \frac{B_{main}}{2} \quad (18)$$

Table III gives the execution time in μsec, for various combinations of the number of PEs (N) and the bandwidth (B_{main}) in $MBps$, as per Equation 17. Figure 5 gives the execution time versus bandwidth curves for various values of N. The curves show that the execution time, as per Equation 17, decreases with the increasing bandwidth, where the execution time is equal to the memory access time, as the computational time is nearly zero.

TABLE III

EXECUTION TIME (T_{exec}) IN μsec FOR VARIOUS COMBINATIONS OF N AND B_{main} IN $MBps$, AS PER EQUATION 17

B_{main} \ N	**500**	**250**	**125**	**100**	**50**	**25**	**20**	**10**	**5**	**4**	**2**	**1**
100	5000	1250	312	200	50	12.5	8	2	0.5	0.32	0.08	0.02
200	2500	625	156	100	25	6.25	4	1	0.25	0.16	0.04	0.01
300	1667	417	104	67	17	4.17	2.7	0.67	0.17	0.1	0.03	0.007
400	1250	312	78	50	12	3.12	2	0.5	0.12	0.08	0.02	0.005
500	1000	250	62	40	10	2.5	1.6	0.4	0.1	0.06	0.016	0.004
600	833	208	52	33	8	2.1	1.3	0.33	0.08	0.05	0.013	0.0033
700	714	178	44	28	7	1.8	1.14	0.28	0.07	0.046	0.011	0.0029
800	625	156	39	25	6.2	1.6	1	0.25	0.06	0.04	0.01	0.0025
900	556	139	34	22	5.6	1.4	0.89	0.22	0.056	0.036	0.0089	0.0022
1000	500	125	31	20	5	1.25	0.8	0.2	0.05	0.032	0.008	0.002

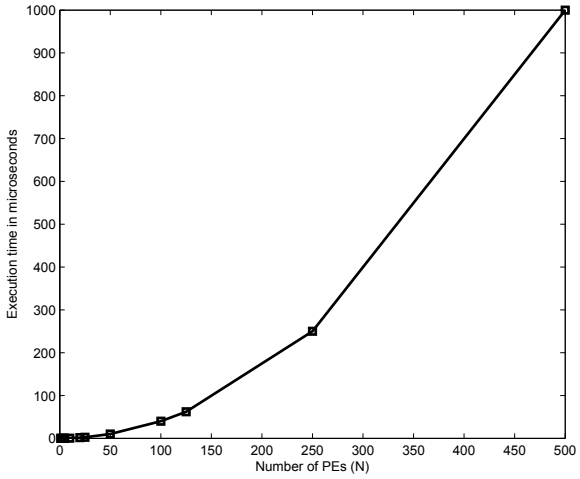

Fig. 6. T_{exec} vs N curve, limited by bandwidth, where T_{exec} increases with the increasing number of PEs (N), for $B_{main} = 500\ MBps$

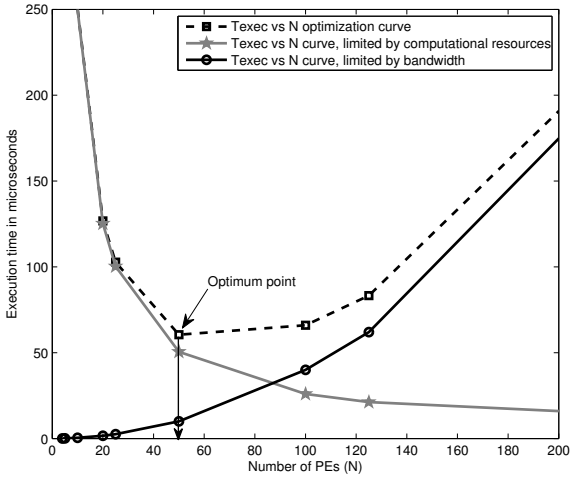

Fig. 7. T_{exec} vs N design trade off curves for $B_{main} = 500\ MBps$ and $T_{PE} = 10\ ns$

For a limited bandwidth, the execution time increases with an increasing length of the query sequence. Figure 6 shows the T_{exec} vs N curve for a case, where the limited bandwidth is 500 $MBps$ and the number of PEs (N) varies from 1 to 500. The execution time, as per Equation 17, has a quadratic dependence on N, which causes it to increase rapidly for higher N values. In the next section, we investigate the minimum execution time that gives optimum performance with reduced bandwidth requirement.

VI. PERFORMANCE AND BANDWIDTH OPTIMIZATION

In this section, performance gain and bandwidth requirements are optimized and a generalized equation is developed for the execution time that considers both the computational resources and bandwidth limitations. Figure 7 shows T_{exec} vs N design trade off curves for the three cases, i.e. when performance is limited by,

- The computational resources
- The bandwidth
- The computational resources and the bandwidth

T_{exec} decreases with the increasing number of N along the T_{exec} vs N curve, limited by the computational resources and based on Equation 15. Decreased T_{exec} results in improved performance, but the bandwidth requirement also increases as a consequence. On the other hand, for a particular available bandwidth, T_{exec} increases with the increasing number of PEs along the T_{exec} vs N curve, limited by the bandwidth and based on Equation 17. The T_{exec} vs N optimization curve represents the total execution time, considering both computational resources and bandwidth limitations and is based on the following equation,

$$T_{exec} = (k \times N_s + N - 1) \times T_{PE} + \frac{2N^2}{B_{main}} \tag{19}$$

To find the N value, at which the function T_{exec} is minimum along the T_{exec} vs N optimization curve, we differentiate Equation 19 w.r.t. N.

$$\frac{d(T_{exec})}{dN} = \frac{d}{dN}\left[(k \times N_s + N - 1) \times T_{PE} + \frac{2N^2}{B_{main}}\right]$$

where $k = \frac{N_q}{N'_q} = \frac{N_q}{N}$, so,

$$\frac{d(T_{exec})}{dN} = \frac{d}{dN}\left[(\frac{N_q \times N_s}{N} + N - 1) \times T_{PE} + \frac{2N^2}{B_{main}}\right]$$
$$= \frac{4N^3 + T_{PE} \times B_{main} \times N^2 - T_{PE} \times B_{main} \times N_q \times N_s}{N^2 \times B_{main}}$$

Now, to find the N value at which T_{exec} is minimum along the T_{exec} vs N optimization curve, we equate $\frac{d(T_{exec})}{dN}$ to

zero, so that,

$$4N^3 + T_{PE} \times B_{main} \times N^2 - T_{PE} \times B_{main} \times N_q \times N_s = 0 \quad (20)$$

The discriminant of Equation 20 is,

$$\Delta = 4T_{PE}^2 \times B_{main}^2 \times N_q \times N_s (T_{PE}^2 \times B_{main}^2 - 108 N_q \times N_s)$$

There are two cases, i.e.

1) $\Delta > 0$, if $N_q \times N_s < \frac{T_{PE}^2 \times B_{main}^2}{108}$, which does not take place in practice. Therefore, this case is not taken into consideration.
2) $\Delta < 0$, if $N_q \times N_s > \frac{T_{PE}^2 \times B_{main}^2}{108}$, which implies that Equation 20 has a unique positive real solution which is given as,

$$N = \frac{A^2 - 3T_{PE} \times B_{main}}{6A} \quad (21)$$

where

$$A = \sqrt[3]{27 T_{PE} B_{main} N_q N_s + 3\sqrt{3T_{PE}^3 B_{main}^3 + 81 T_{PE}^2 B_{main}^2 N_q^2 N_s^2}}$$

For a given bandwidth, Equation 21 gives the N value at which the function T_{exec} is minimum along the T_{exec} vs N optimization curve. The minimum T_{exec} value guarantees an optimum performance, as any performance gain due to increasing number of PEs beyond this point is counterbalanced by the bandwidth limitation. As an example, if,

$$T_{PE} = 10\ ns,\ B_{main} = 500\ MBps \text{ and } N_q = N_s = 500,$$

then, the value of N, as per Equation 21, would be $N = 67.8$. This means that $N = 68$ guarantees an optimum performance for the given example. Therefore, any further increase in the number of PEs will result in a subsequent performance loss due to bandwidth limitation.

Figure 8 shows the optimization curves for various values of B_{main} in $MBps$, where the optimum point shifts towards higher N values for increasing bandwidth. This implies that for higher available bandwidth, a higher value of N can be chosen to improve the performance further.

VII. Conclusion

In this paper, we presented the performance and bandwidth analysis of biological sequence alignment and formulated theoretical performance boundaries for the two cases, when performance is limited by,

- The computational resources
- The bandwidth

These two cases were further used to develop a generalized equation for the execution time, when both limitations were in effect. The results demonstrated a performance improvement with the increasing number of PEs due to increased parallelism, but the bandwidth requirement also increased as a consequence, limiting the overall performance gain. Moreover, we optimized the number of PEs for a minimum execution time, taking the available bandwidth into consideration. The minimum execution time guaranteed an optimum performance, as any performance gain due to increased number of PEs was counterbalanced by the bandwidth limitation beyond this point.

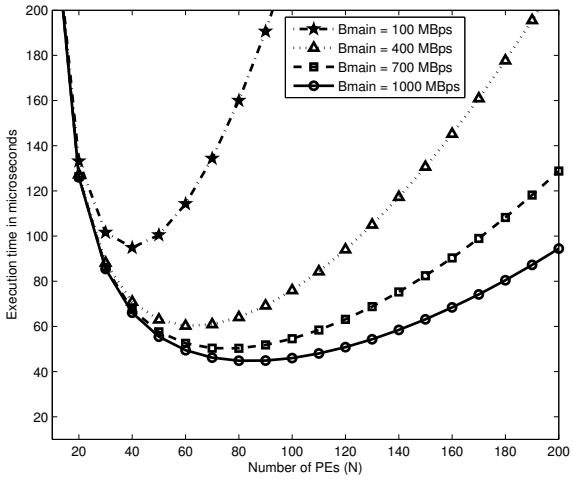

Fig. 8. T_{exec} vs N optimization curves for $T_{PE} = 10\ ns$ and various values of B_{main} in $MBps$

References

[1] M. Vingron and M. S. Waterman, "Sequence Alignment and Penalty Choice: Review of Concepts, Case Studies and Implications", *Journal of Molecular Biology*, vol. 235, pages 1–12, 1994.

[2] A. YarKhan and J. J. Dongarra, "Biological Sequence Alignment on the Computational Grid Using the GrADS Framework", *Future Generation Computer Systems*, vol. 21(6), pages 980–986, June 2005.

[3] L. Hasan, Z. Al-Ars and S. Vassiliadis, "Hardware Acceleration of Sequence Alignment Algorithms - An Overview", *Proceedings of International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS'07)*, pages 96–101, Rabat, Morocco, September 2–5, 2007.

[4] S. F. Altschul et al., "A Basic Local Alignment Search Tool", *Journal of Molecular Biology*, vol. 215, pages 403–410, 1990.

[5] W. R. Pearson and D. J. Lipman, "Rapid and Sensitive Protein Similarity Searches", *Science*, vol. 227, pages 1435–1441, 1985.

[6] S. R. Eddy, "Profile Hidden Morkov Models", *Bioinformatics Review*, vol. 14(9), pages 755–763, July 1998.

[7] R. Giegerich, "A Systematic Approach to Dynamic Programming in Bioinformatics", *Bioinformatics*, vol. 16, pages 665–677, 2000.

[8] T. F. Smith and M. S. Waterman, "Identification of Common Molecular Subsequences", *Journal of Molecular Biology*, vol. 147, pages 195–197, 1981.

[9] L. Hasan and Z. Al-Ars, "An Efficient and High Performance Linear Recursive Variable Expansion Implementation of the Smith-Waterman Algorithm", *31st Annual International Conference of the IEEE EMBS*, pages 3845–3848, Minneapolis, Minnesota, USA, September 2009.

[10] A. B. Buyukkur and W. Najjar, "Compiler Generated Systolic Arrays for Wavefront Algorithm Acceleration on FPGAs", *International Conference on Field Programmable Logic and Applications (FPL08)*, Heidelberg, Germany, September 2008.

[11] A. D. Blas et al., "The UCSC Kestrel Parallel Processor", *IEEE Transactions on Parallel and Distributed Systems*, vol. 16(1), pages 80–92, 2005.

[12] A. Schroder et al., "Bio-Sequence Database Scanning on a GPU" *HICOMB*, 2006.

[13] L. Hasan, Z. Al-Ars, Z. Nawaz and K. L. M. Bertels, "Hardware Implementation of the Smith-Waterman Algorithm Using Recursive Variable Expansion", *Proceedings of 3rd Inernational Design and Test Workshop IDT08*, Monastir, Tunisia, December 2008.

[14] T. Oliver, B. Schmidt and D. Maskell, "Hyper Customized Processors for Bio-Sequence Database Scanning on FPGAs", *FPGA'05*, Monterey, California, USA, February 20–22, 2005.

[15] Boeckmann et al., "The SWISS-PROT Protein Knowledge Base and its Supplement TrEMBL", *Nucleic Acids Research*, vol. 31, pages 365–370, 2003.